Projekt

Teoria i metody optymalizacji

Harmony Search

Autor:
Artur Gasiński, 218685
Bartosz Lenartowicz, 218518

Termin: wt 9:15-11:00

Prowadzący:
Dr inż. Ewa Szlachcic

27 maja 2018

Spis treści

1 Sformułowanie zadania optymalizacji

Większość metod optymalizacji procesów produkcyjnych czy logistycznych sprowadza się do przedstawienia zależności pomiędzy procesami za pomocą okrelonej funkcji celu *f(x)*. Pod postacią zmiennych x tego równania ukazane są parametry procesów, którymi można manipulować. By właściwie zoptymalizować proces, należy znaleźć minimum bądź maksimum (najlepiej globalne) funkcji *f(x)=f*(x*)*. Obliczanie minimalnej bądź maksymalnej wartości *f**, zależnej od wielu zmiennych x w pożądanym czasie jest problematyczne. Istnieją algorytmy specjalizujące się w rozwiązywaniu zadań tego typu. Niniejsza praca została napisana by przedstawić jeden z nich – Harmony Search. Został on przedstawiony w 2001 przez Zong Woo Geema, Joong Hoon Kima oraz G.V. Loganathana w pracy „A New Heuristic Optimization Algorithm: Harmony Search" [1].

2 Szczegółowe omówienie algorytmu optymalizacji

Inspiracją dla algorytmu było szukanie przez muzyków jazzowych najlepszych harmonii dźwięków podczas improwizacji. Jego matematyczny odpowiednik umożliwia optymalizację funkcji *f(x)* wielu zmiennych.

Algorytm do działania wymaga zainicjowania $m = 4 + 2 * n$ wartości początkowych. Jedynym kryterium stopu algorytmu jest liczba iteracji L. Głównym elementem algorytmu jest pamięć harmonii `HM` (*ang. harmony memory*). Jest to zbiór zapamiętanych rozwiązań funkcji $f(x)$ wraz z wartościami argumentów x, dla których została obliczona funkcja. Schemat tablicy `HM` umieszczony jest na rysunku 1. Wielkość pamięci *HMS* deklarowana jest z góry przez użytkownika i podczas kolejnych iteracji nie zmienia rozmiaru. Wymagane jest również by na początku działania określić przedział wartości $[a, b]$ dla każdej zmiennej $x_1 = [a_1, b_1], ..., x_n = [a_n, b_n]$, z którego będą wyszukiwane rozwiązania dlatego należy zadeklarować $2 * n$ wartości gdzie n jest ilością zmiennych równania. Dodatkowo wymagane jest określenie dwóch prawdopodobieństw. Pierwsze `HMCR` (*ang. harmony memory consideration ratio*) określa współczynnik doboru tonu z pamięci, drugie `PAR` (*ang. pitch adjustment ratio*) nazywane jest współczynnikiem dostosowania tonu. Oczywistym jest, że przedział wartości prawdopodobieństw określany jest na $< 0, 1 >$.

Algorytm działa następująco: wpierw program wypełnia tablicę pamięci harmonii `HM` początkowymi wartościami. Każda wartość x losowana jest z swojego ograniczonego przedziału $[a, b]$. Gdy tablica zostanie zapełniona rozpoczyna się część iteracyjna algorytmu. Polega na wyszukiwaniu kolejnego rozwiązania poprzez wyznaczanie kolejnych rozwiązań wedle trzech reguł. Pierwsza mówi, że z prawdopodobieństwem $P_1 =$ `HMCR` wartość nowej zmiennej x_n wybierana jest z pośród zmiennych zapamiętanych w tablicy `HM`. Druga, że wartością prawdopodobieństwa $P_2 = (1 -$ `HMCR`$)$ wartość zmiennej losowana jest z całego przedziału $[a, b]$, a zasada trzecia, że zmienna ustalona w regule pierwszej modyfikowana jest z prawdopodobieństwem $P_3 =$ (`HMCR` $*$ `PAR`). W praktyce reguła trzecia sprowadza się do dodania do wylosowanej zmiennej x z reguły pierwszej liczby z przedziału $< -c, c >$. Następnie gdy zostaną wylosowane wszystkie wartości x_n obliczana jest dla nich wartość funkcji $f(x)$. Jest ona porównywana z wartościami umieszczonymi w tablicy `HM`. Jeżeli jest mniejsza niż największa wartość z tablicy, zamienia się ją z największą wartością (najgorszym rozwiązaniem). Na tym kończy się iteracja. Następnie wykonuje się kolejne iteracje do osiągnięcia maksymalnej ilości iteracji L. Polski opis algorytmu można również znaleźć np. w pracy [2].

x_{11}	$\cdots$	x_{n1}	$= f_1$
$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{1HMS}	$\cdots$	x_{nHMS}	$= f_{HMS}$

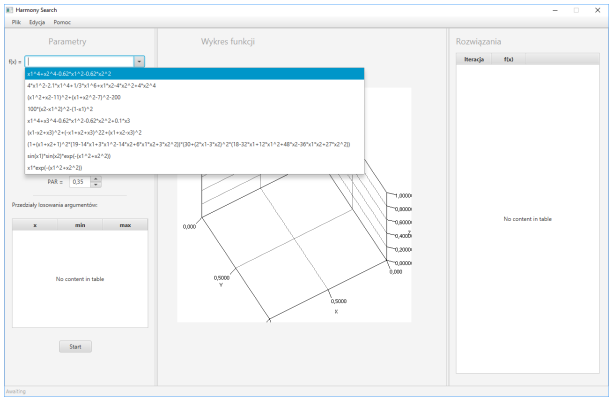
Rysunek 1: Schemat tabeli *Harmony Memory*

3 Informacje ogólne o programie

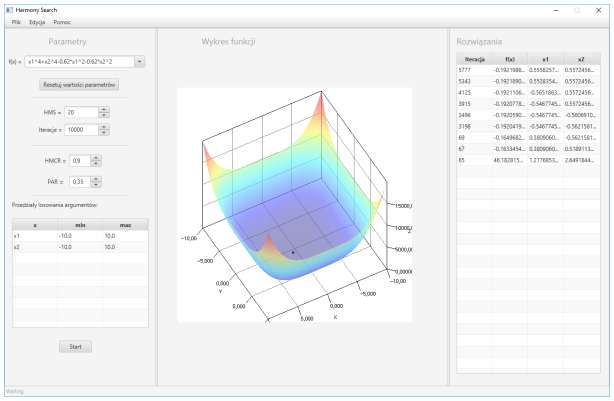
Program do obliczania minimum funkcji na podstawie algorytmu Harmony Search został napisany w języku Java. Technologią dostarczającą GUI dla użytkownika była *JavaFx* umożliwiająca tworzenie aplikacji okienkowych. Dodatkowo by ułatwić zarządzanie bibliotekami wewnątrz projektu użyto wsparcia ze strony *Maven*. Program korzysta z zewnętrznej biblioteki mXparser do parsowania łańcuchów znakowych na funkcje i do obliczania ich wartości w określonych punktach [3]. Wykres prezentujący najlepsze rozwiązanie zaznaczone na wykroju z przestrzeni rozwiązań funkcji został stworzony za pomocą biblioteki *jzy3D*.

4 Środowisko programistyczne, zastosowane biblioteki i gotowe moduły

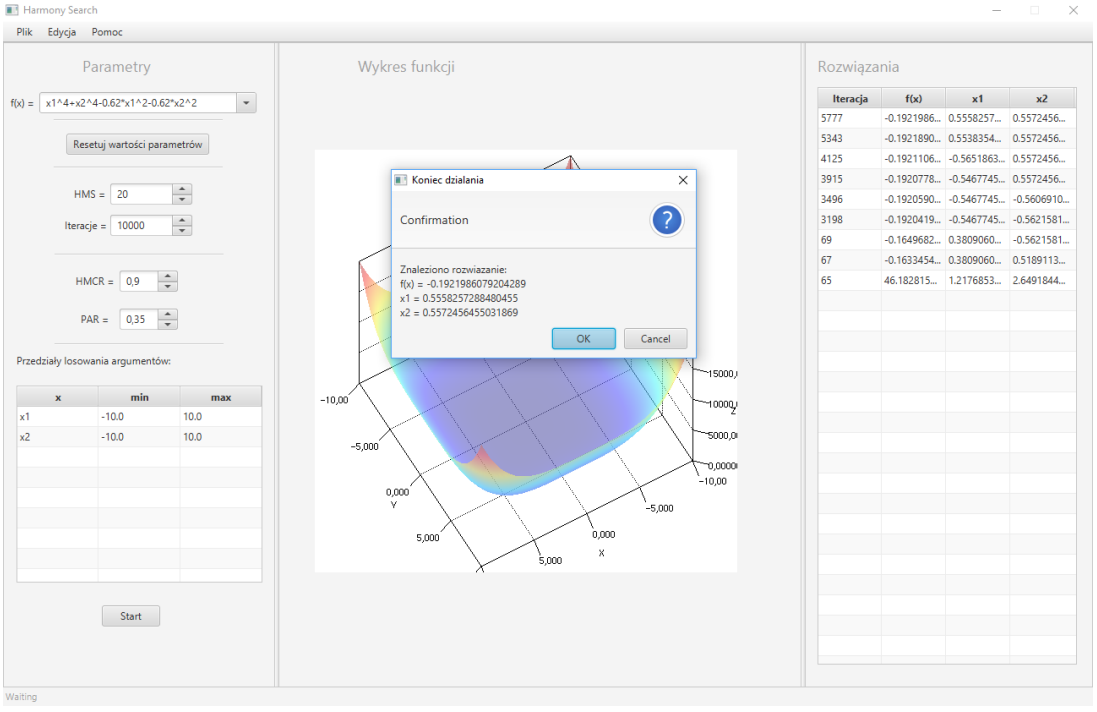
By ułatwić wprowadzanie funkcji do programu oraz bezproblemowo odczytywać wyniki obliczone przez program zostało stworzone GUI programu. Na rysunkach 2, 3 i 4 przedstawiono etapy działania programu. Poszczególne elementy GUI podczas działania programu nie zmieniają swojego położenia. Jedynie pustych w miejscach zostają wyświetlone ich parametry, dlatego by omówić elementy posłużono się rysunkiem 4, gdzie widnieje obraz programu po obliczeniach. Główne okno programu zostało podzielone pionowo na trzy części. Pierwsza część od lewej przeznaczona jest do wpisania funkcji i jej parametrów, środkowa prezentuje wykres, a część prawa w tabelce wyniki obliczeń programu. Wszystkie części zostaną szczegółowo opisane poniżej.



Rysunek 2: Wpisywanie funkcji



Rysunek 3: Wykres z zaznaczonym rozwiązaniem



Rysunek 4: Obliczone rozwiązanie wraz z komunikatem

5 Parametry

W części parametrów można dostrzec, że zaraz pod paskiem edycji znajduje miejsce do wpisania funkcji, której minimum należy wyznaczyć. Okno umożliwia również wybór takiej funkcji z wcześniej zadeklarowanych w programie. Dokonuje się tego, klikając w rozwijane menu. Zostało to przedstawione na rysunku 2. Po wybraniu funkcji automatycznie zostaną domyślnie uzupełnione wszystkie parametry obliczeń. Oczywiście możliwa jest dostosowanie algorytmu do własnych potrzeb. Poniżej okna funkcji znajduje się przycisk „`Resetuj wartości parametrów`" umożliwiający, jak mówi nazwa przywrócenie domyślnych wartości parametrów. Poniżej przycisku zostały umieszczone po sobie cztery główne parametry algorytmu *Harmony Search*.

Pierwszy z parametrów `HMS` (*ang. harmony memory size*) reguluje ilość zapamiętywanych rozwiązań podczas wyszukiwania. Domyślnie rozmiar jest zadeklarowany na 20. Poniżej znajduje się okno o nazwie `iteracje`. Domyślnie jest zadeklarowana maksymalna wartość iteracji równa 10000. W każdej iteracji programu algorytm wyszukuje nowe rozwiązanie i sprawdza je z tymi zapisanymi w `HMS`. Im większa liczba iteracji, tym algorytm powinien obliczać dokładniejsze rozwiązanie, kosztem wydłużonego czasu obliczeń. Następnie pod polem `iteracje` znajduje się pole `HMCR` (*ang. harmony memory consideration ratio*). Jest to prawdopodobieństwo wyboru rozwiązania z pamięci (z zakresu $(0, 1)$). Im `HMCR` jest większe tym wyszukiwane wartości x następnych rozwiązań będą zbliżone do tych istniejących już w tablicy `HMS`. Poniżej znajduje się drugi parametr prawdopodobieństwa o nazwie `PAR` *(ang. pitch adjustment ratio)* tłumaczony jako współczynnik dostosowywania tonu. Wybierany jest również z zakresu $(0, 1]$. Na samym dole kolumny znajduje się tabela `Przedziały losowania argumentów`. W tabeli deklarowany jest przedziały wszystkich wartości x, z których zostaną wylosowane początkowe argumenty podczas inicjalizacji pamięci harmonii. Domyślnie parametry dla wszystkich x brane są z przedziału $(-10, 10)$. Algorytm z dużą dokładnością znajdzie najmniejsze minimum lokalne dla funkcji z podanego przedziału. Po wybraniu wszystkich parametrów można przystąpić do uruchomienia algorytmu przyciskiem `Start` znajdującym się na na dole kolumny.

5.1 Wykres

Środkowy obszar okna głównego przeznaczony jest do wyświetlania wykresu funkcji. Jak wspomniane zostało na początku rozdziału 3, do rysowania wykresów została użyta zewnętrzna biblioteka *jzy3D*. Umożliwia ona tworzenie wykresów dwu- i trójwymiarowych. Wykresy można obracać w dowolny sposób. Oś X i Y wykresu wyskalowana jest do zgodnie z podanymi przedziałami losowania argumentów x. Dodatkowo wartości zostały pokolorowane w tak, że mniejsze wartości funkcji mają kolory zimniejsze, a wyższe - kolory cieplejsze, analogicznie do tego jak przedstawia się wysokość n.p.m. w kartografii. Dodatkowo czarną kropką zaznaczona została najmniejsza obliczona wartość funkcji.

5.2 Tabela rozwiązań

Po prawej stronie okna głównego można dostrzec najlepsze rozwiązania wyszukane przez program. Gdy znajdowane jest rozwiązanie lepsze od najlepszego rozwiązania zapamiętanego w `HMS` dodawane jest ono jako pierwszy wiersz w tabeli. Poprzednie rozwiązania przesuwane są o jedno miejsce niżej. Pozwala to śledzić jaka wartości jest aktualnie największa oraz od razu ukazuje się obok numer iteracji tego rozwiązania. Gdy algorytm długo oblicza najlepszy wynik, okno pozwala śledzić postęp obliczeń. Dodatkowo po wykonaniu wszystkich iteracji zostaje wyświetlony komunikat z najlepszym rozwiązaniem (pierwszym od góry), co przedstawia rysunek 4.

6 Przykłady rozwiązań oraz testy

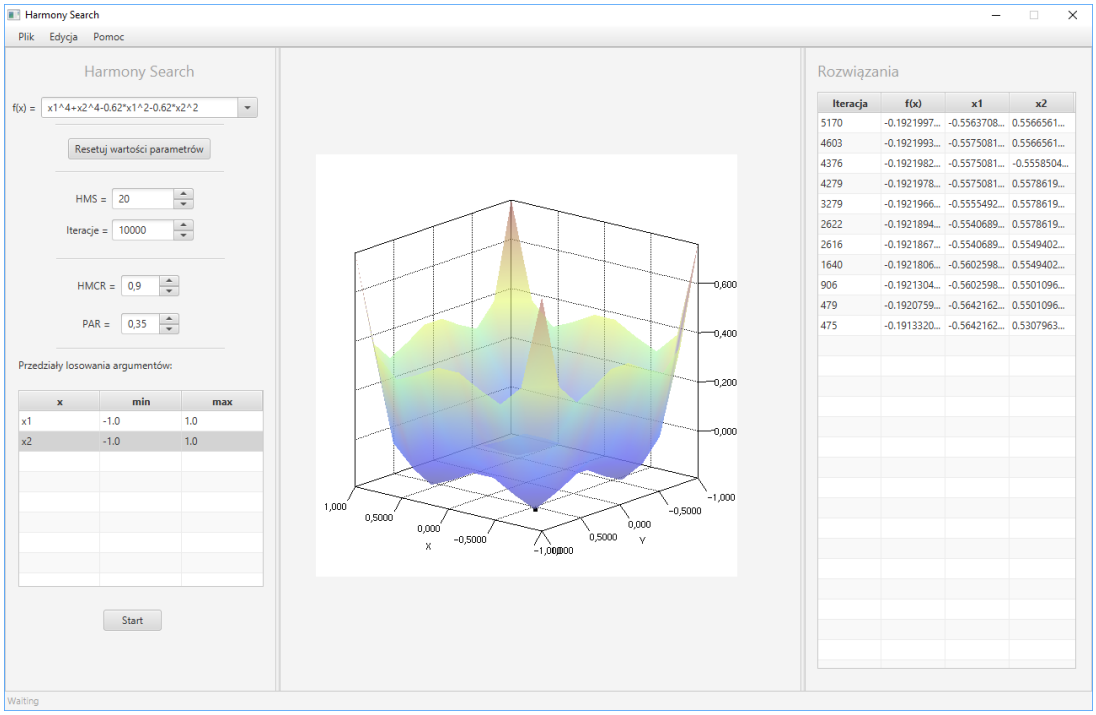
W tej części zostały przedstawione przykłady funkcji wraz z wynikami obliczonymi dzięki przedstawianemu programowi.

6.1 Funkcja czterech minimów

Pierwsza zostanie przedstawiona funkcja posiadająca cztery minima lokalne. Wyraża się ona wzorem:

$$f(x_1, x_2) = x_1^4 + x_2^4 - 0.62x_1^2 - 0.62x_2^2.$$

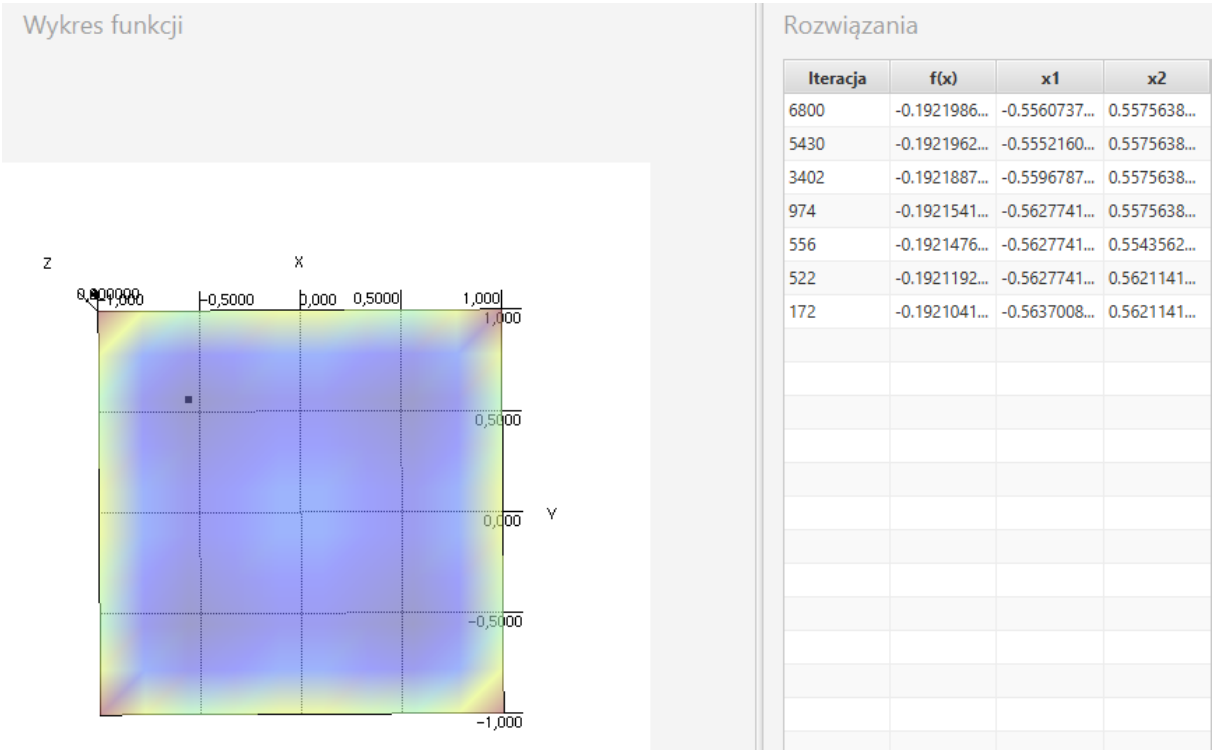
Rysunek 5 prezentuje trójwymiarowy wykres funkcji na przedziale $x_1, x_2 \in < -1, 1 >$ wraz z zaznaczoną najmniejszą obliczoną wartością $f(x^*) = -0.19219, x^*(i) = [\pm 0.55672; \pm 0.55672], i = 1, ..., 4$.



Rysunek 5: Wykres funkcji czterech minimów $x_1, x_2 \in < -1, 1 >$

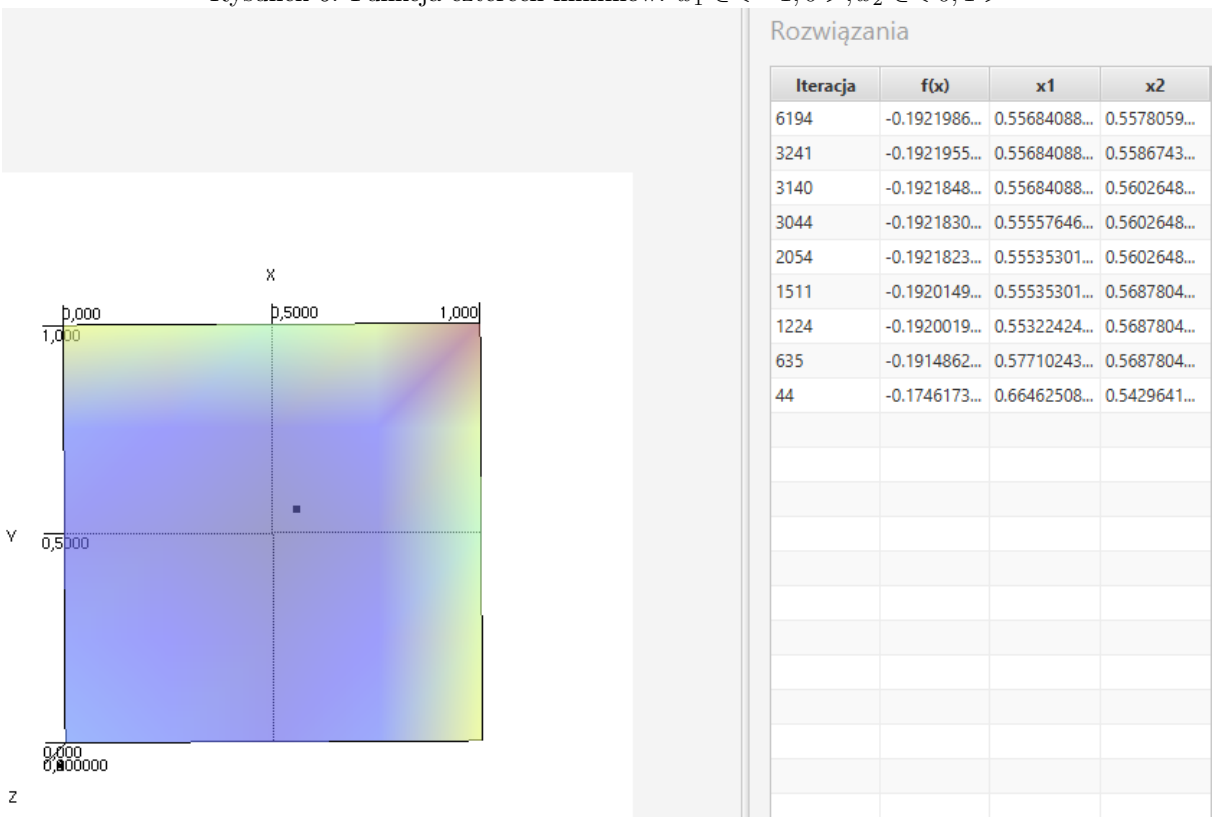
6.1.1 Wyznaczenie wszystkich minimów

By wyznaczyć wszystkie minima funkcji należało zawęzić przedziały losowania argumentów w taki sposób, by w każdym przedziale było jedno minimum lokalne. Przypadki prezentowane na rysunkach 6.1.1, kolejno pokazują każde obliczone minimum zależnie od podanego przedziału. Jak możemy dostrzec na wykresach algorytm znalazł poprawne rozwiązanie w każdym przedziale. Wartości minimum zaznaczone są na wykresach w postaci czarnej kropki oraz wpisane są w najwyższy wiersz tabeli `Rozwiązania` pokazanej po prawej stronie.



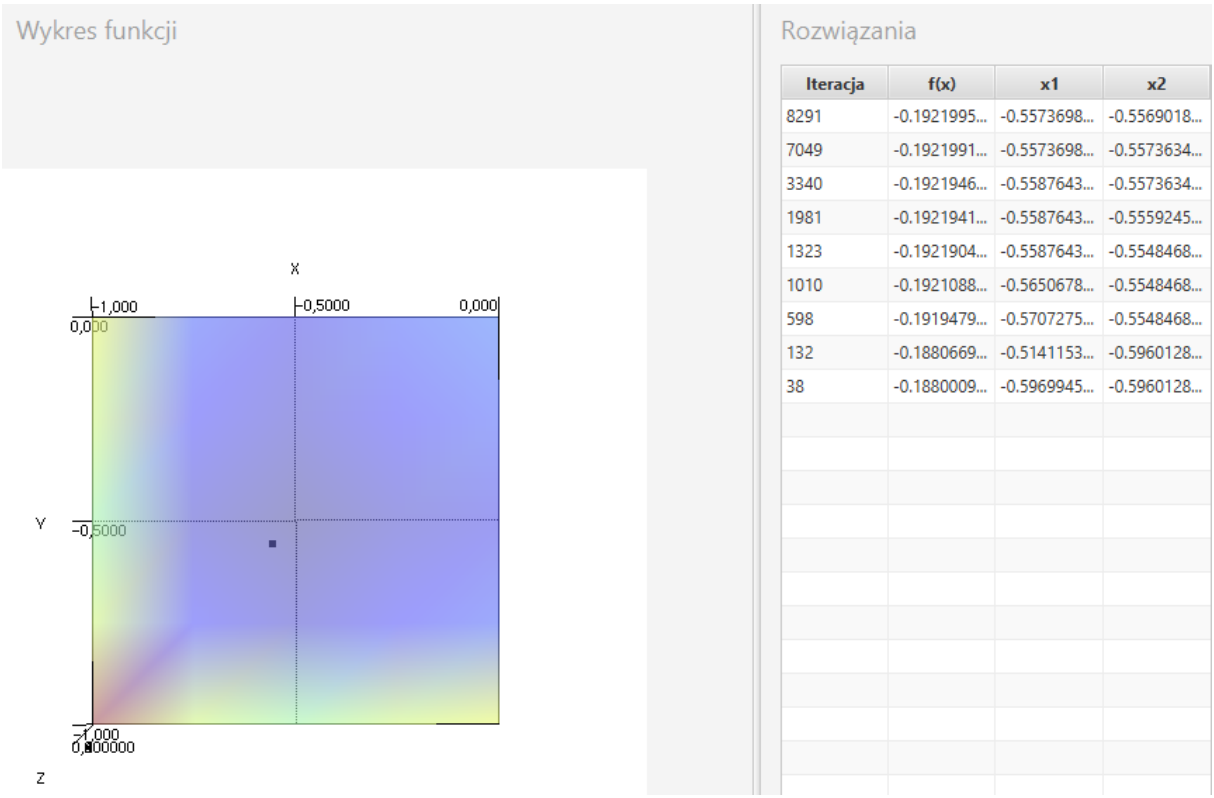
Iteracja	f(x)	x1	x2
6800	-0.1921986...	-0.5560737...	0.5575638...
5430	-0.1921962...	-0.5552160...	0.5575638...
3402	-0.1921887...	-0.5596787...	0.5575638...
974	-0.1921541...	-0.5627741...	0.5575638...
556	-0.1921476...	-0.5627741...	0.5543562...
522	-0.1921192...	-0.5627741...	0.5621141...
172	-0.1921041...	-0.5637008...	0.5621141...

Rysunek 6: Funkcja czterech minimów: $x_1 \in < -1, 0 >, x_2 \in < 0, 1 >$

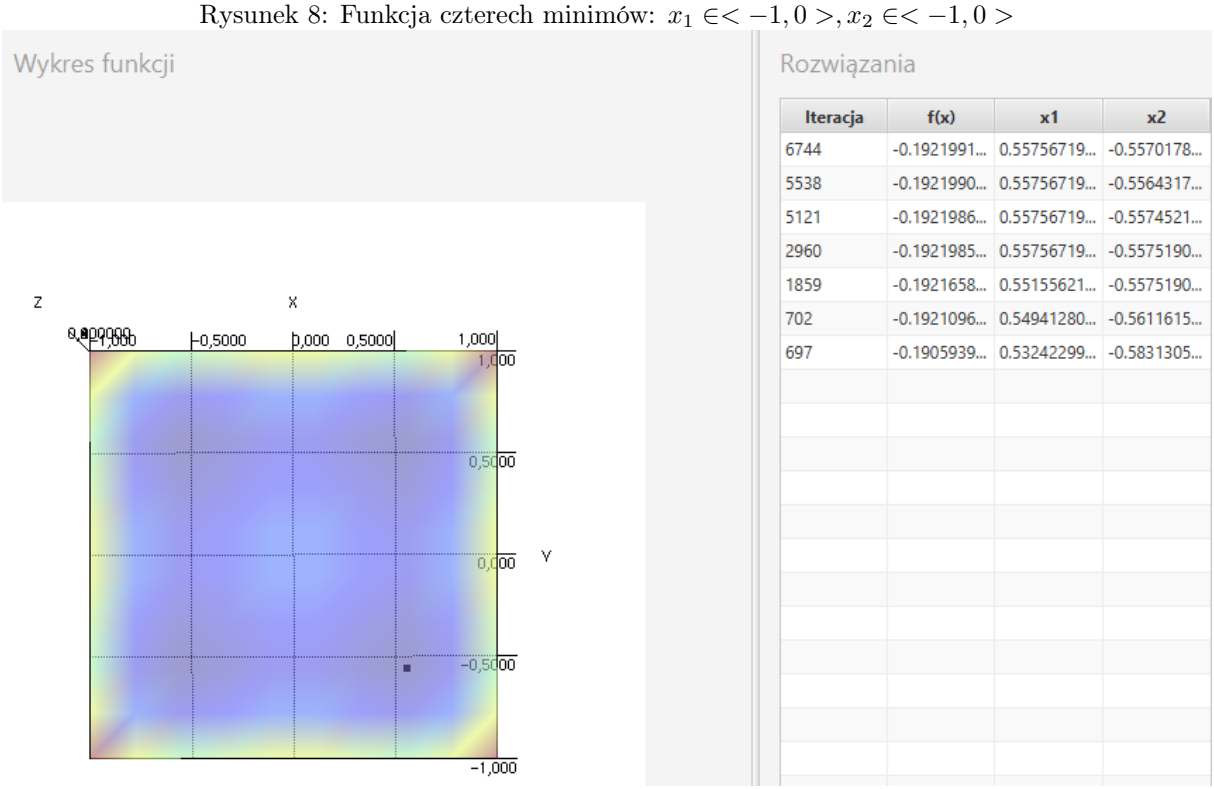


Iteracja	f(x)	x1	x2
6194	-0.1921986...	0.55684088...	0.5578059...
3241	-0.1921955...	0.55684088...	0.5586743...
3140	-0.1921848...	0.55684088...	0.5602648...
3044	-0.1921830...	0.55557646...	0.5602648...
2054	-0.1921823...	0.55535301...	0.5602648...
1511	-0.1920149...	0.55535301...	0.5687804...
1224	-0.1920019...	0.55322424...	0.5687804...
635	-0.1914862...	0.57710243...	0.5687804...
44	-0.1746173...	0.66462508...	0.5429641...

Rysunek 7: Funkcja czterech minimów: $x_1 \in < 0, 1 >, x_2 \in < 0, 1 >$



Rysunek 8: Funkcja czterech minimów: $x_1 \in < -1, 0 >, x_2 \in < -1, 0 >$



Rysunek 9: Funkcja czterech minimów: $x_1 \in < 0, 1 >, x_2 \in < -1, 0 >$

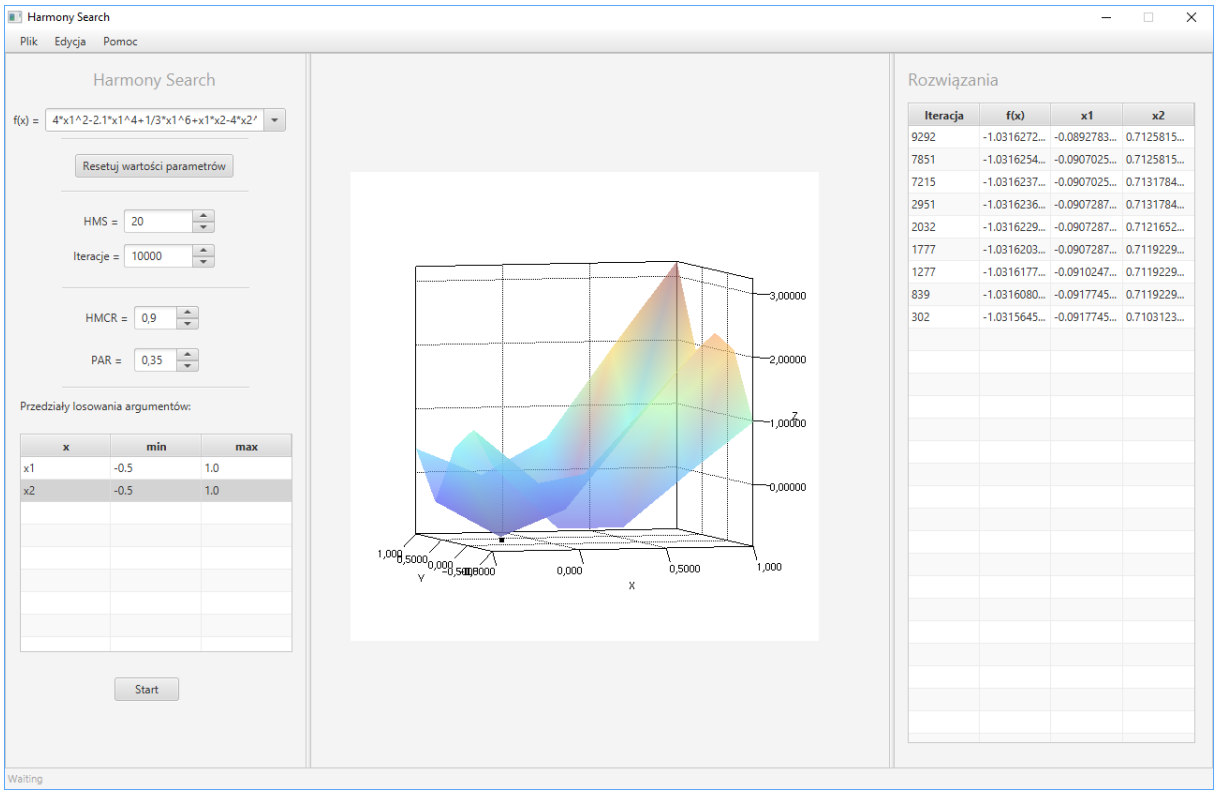
Rysunek 10: Wykres funkcji czterech minimów dla różnych przedziałów

6.2 Funkcja testowa Geema

Kolejny przykład przedstawiał funkcję opisaną wzorem:

$$f(x_1, x_2) = x_1^4 + x_2^4 - 0.62x_1^2 - 0.62x_2^2.$$

Jest to funkcja zwana szeciogarbnym wielbłądem: ma 6 minimów lokalnych, w tym 2 globalne: $f(x^*) = f(x^{**} = 1.0316285, x^* = [0.08984, -0.71266], x^{**} = [-0.08984, 0.71266]$. Początkowo zakres dla argumentów funkcji został przyjęty jako $x_1, x_2 \in < -0.5, 1 >$. Wykres funkcji wraz z rozwiązaniem zaprezentowany jest na rysunku 11.

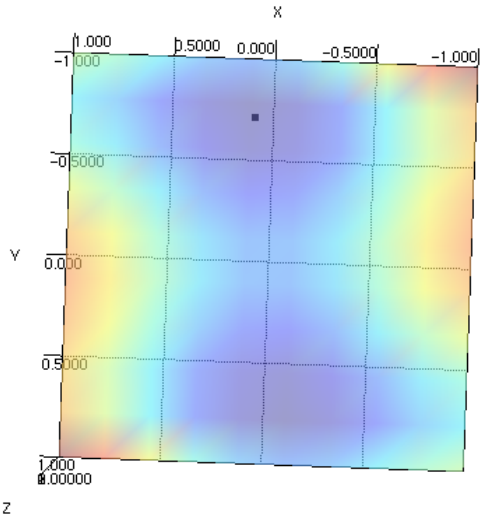


Rysunek 11: Wykres funkcji Geema dla $x_1, x_2 \in < -0.5, 1 >$

6.2.1 Zmiana zakresu dla argumentów

Jak w poprzednim przykładzie tak i tutaj postanowiono zmienić zakres wyszukiwania argumentów na $x_1, x_2 \in < -1, 1 >$. Zostały znalezione obydwa globalne minima.

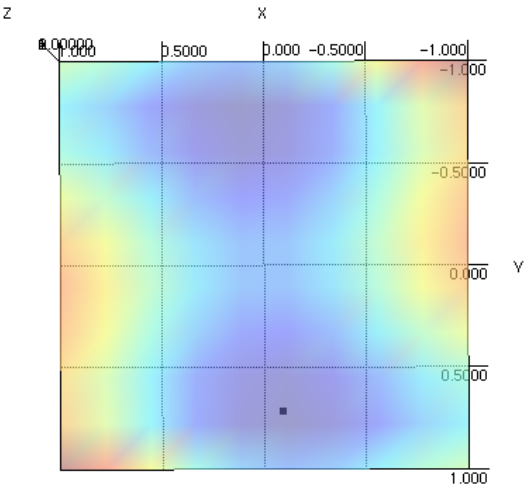
Wykres funkcji



Rozwiązania

Iteracja	f(x)	x1	x2
8906	-1.0316283	0.089754745	-0.7125227
8150	-1.0316185	0.088237286	-0.7125227
7112	-1.031618	0.088237286	-0.7123149
5331	-1.0316135	0.09169863	-0.7123149
2443	-1.0316079	0.09204542	-0.7123149
1531	-1.0315852	0.086725935	-0.7133008
1068	-1.0315784	0.093386754	-0.7133008
825	-1.031559	0.094041854	-0.7133008
250	-1.0288311	0.06321709	-0.7133008

Wykres funkcji



Rozwiązania

Iteracja	f(x)	x1	x2
17774	-1.0316277	-0.08981851	0.71233404
6825	-1.0316275	-0.0899064...	0.71233404
5870	-1.0316273	-0.08953823	0.71233404
5274	-1.0316101	-0.08831959	0.71363413
3591	-1.0315871	-0.0929013...	0.71363413
2747	-1.0315701	-0.0929013...	0.7144856
779	-1.0315467	-0.0929013...	0.7151958
298	-1.0311145	-0.10105913	0.7151958

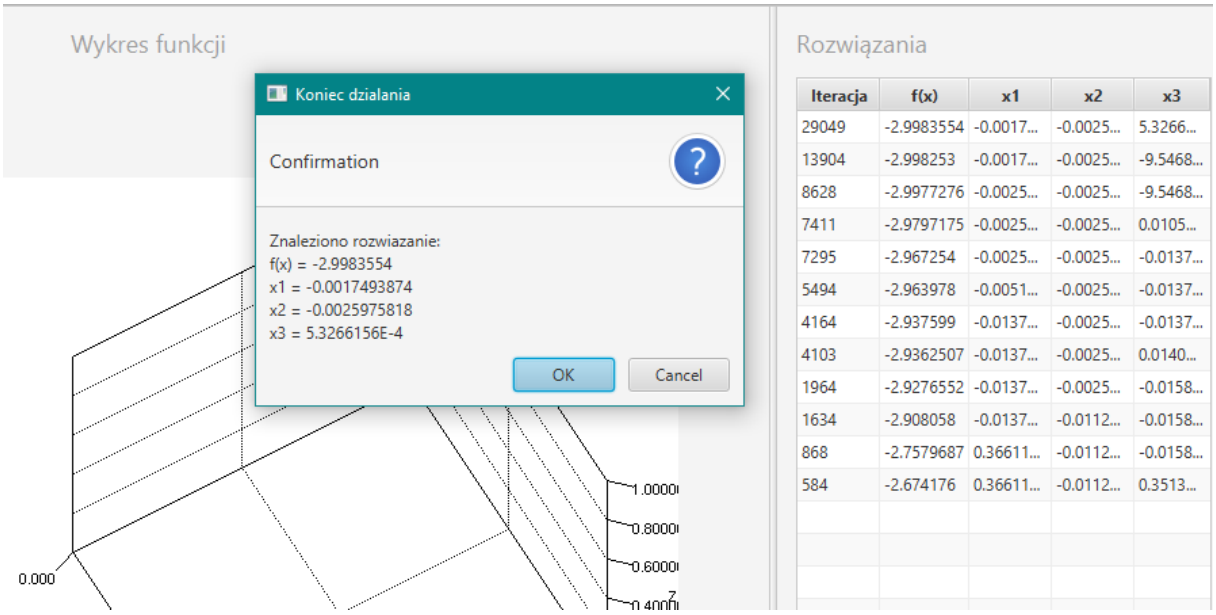
Rysunek 12: Wykresy funkcji Geema dla $x_1, x_2 \in < -1, 1 >$

6.3 Funkcja Rastrigina dla $n = 3$

Funkcja Rastrigina jest określona następująco:

$$f(x) = \sum_{i=1}^{n} x_i^2 - cos(18x_i), x_i \in < -1, 1 > .$$

Ma ona jedno globalne minimum $f(x^*) = -n, x^* = 0$. W poniższym przypadku przyjęto $n = 3$. Program odnajduje je bezproblemowo, chociaż im więcej iteracji, tym bardziej przybliża się do rzeczywistej wartości minimum.



Iteracja	f(x)	x1	x2	x3
29049	-2.9983554	-0.0017...	-0.0025...	5.3266...
13904	-2.998253	-0.0017...	-0.0025...	-9.5468...
8628	-2.9977276	-0.0025...	-0.0025...	-9.5468...
7411	-2.9797175	-0.0025...	-0.0025...	0.0105...
7295	-2.967254	-0.0025...	-0.0025...	-0.0137...
5494	-2.963978	-0.0051...	-0.0025...	-0.0137...
4164	-2.937599	-0.0137...	-0.0025...	-0.0137...
4103	-2.9362507	-0.0137...	-0.0025...	0.0140...
1964	-2.9276552	-0.0137...	-0.0025...	-0.0158...
1634	-2.908058	-0.0137...	-0.0112...	-0.0158...
868	-2.7579687	0.36611...	-0.0112...	-0.0158...
584	-2.674176	0.36611...	-0.0112...	0.3513...

Rysunek 13: Wykres funkcji Rastrigina i znalezione minimum globalne

6.4 Funkcja Himmelblau

Funkcja Himmelbalu również jak 6.1 posiada cztery minima lokalne zlokalizowane po jednym w każdej ćwiartce na przedziale $x_{1,2} \in (-5, 5)$. Funkcja posiada wzór:

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 - 200.$$

Dla pokazania, że funkcja wyszukuje rozwiązania z innego przedziału jak domyślny ograniczono się do przedziału z pierwszej ćwiartki $x_1, x_2 \in (0, 10)$. Na wykresie 6.4 zaprezentowany został wykres funkcji wraz z naniesionym najlepszym rozwiązaniem.

6.5 Funkcja Goldsteina-Price'a z czterema minimami lokalnymi

Funkcja Goldsteina-Price'a posiada kilka minimów lokalnych umiejscowionych blisko siebie. Opisuje ją wzór:

$$f(x_1, x_2) = [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)][30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$$

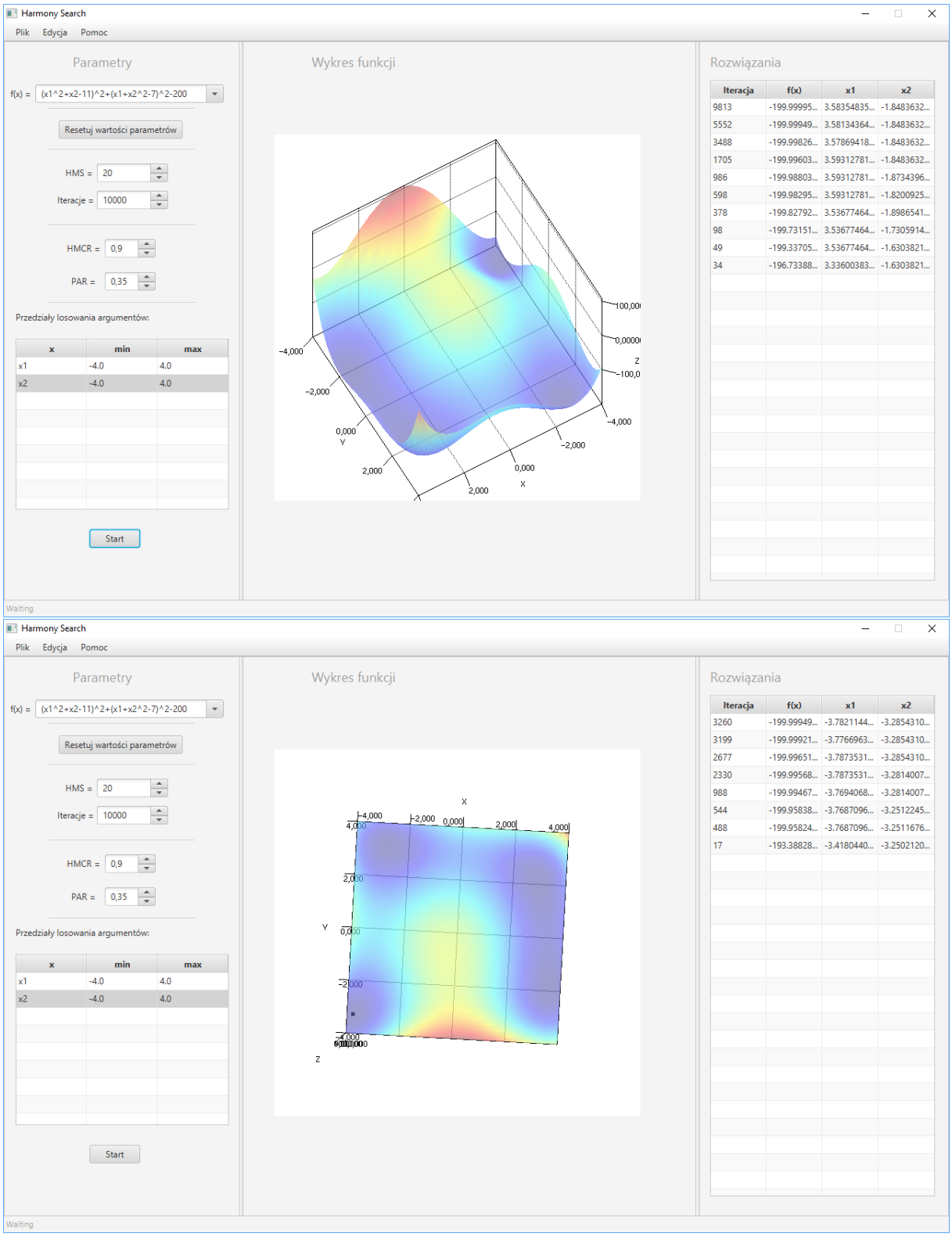
Odpowiednio manewrując parametrami algorytmu, program był w stanie obliczyć wszystkie 4 minima funkcji. Na wykresach 6.5 oraz 6.5 pokazano wyznaczone rozwiązania.

6.6 Dwuwymiarowa funkcja sinusoidalna

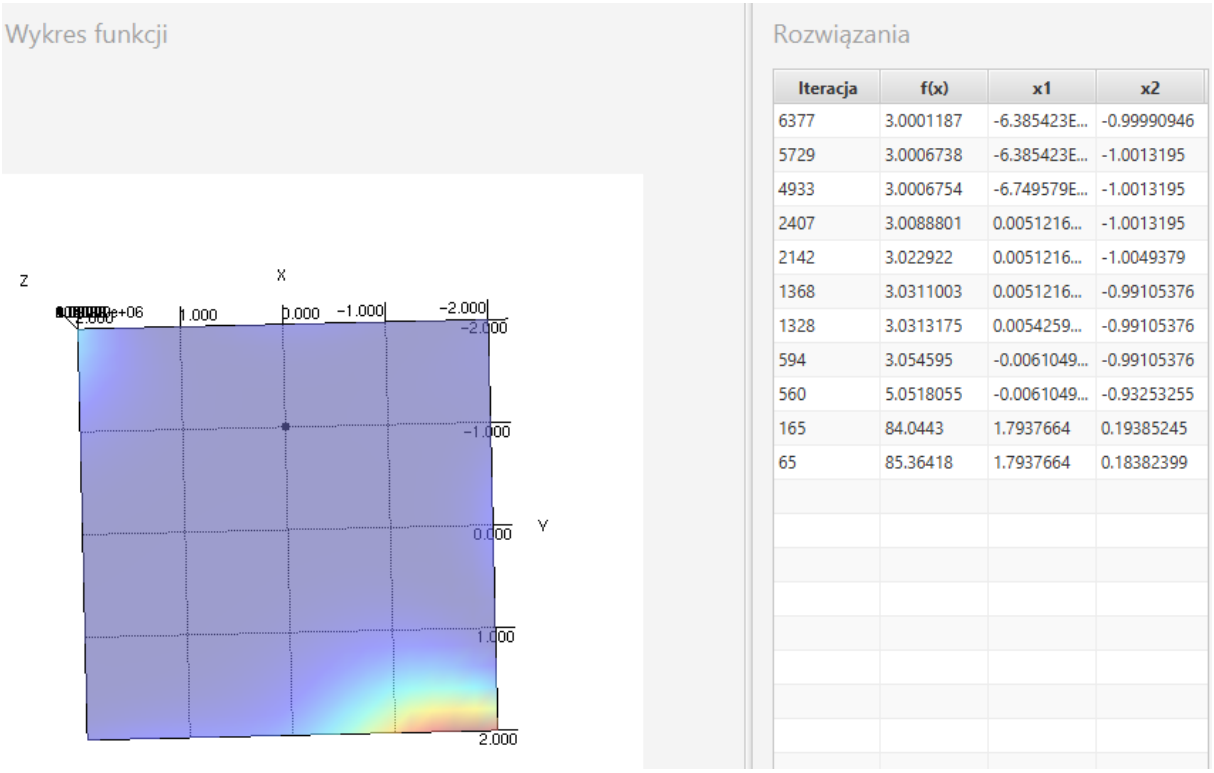
Dwuwymiarowa funkcja sinus posiada bardzo zróżnicowany przebieg oraz wiele równoważnych minimów lokalnych. Wzór funkcji przedstawiony jest równaniem

$$f(x_1, x_2) = \sin x_1 \sin x_2 e^{-(x_1^2+x_2^2)}.$$

Rysunek 6.6 prezentuje wykres funkcji dla argumentów z przedziału $< -1, 1 >$.

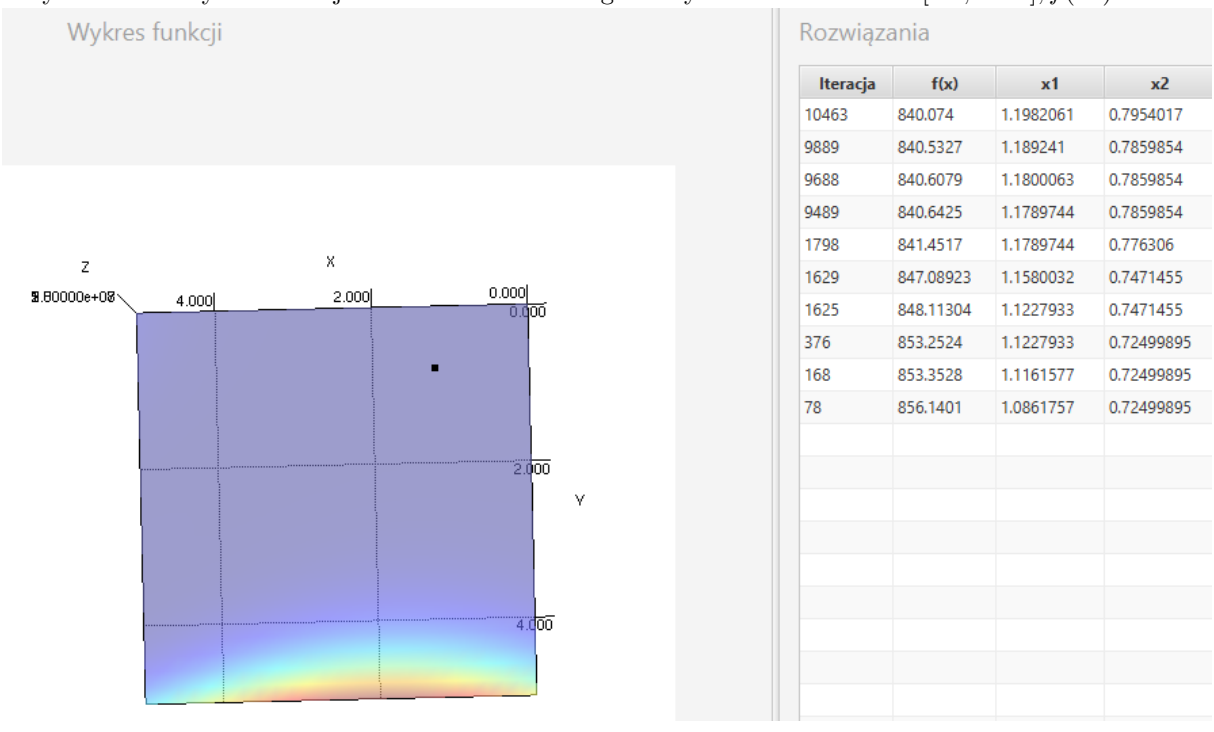


Rysunek 14: Wykresy funkcji Himmelblau dla $x_1, x_2 \in < -4, 4 >$



Iteracja	f(x)	x1	x2
6377	3.0001187	-6.385423E...	-0.99990946
5729	3.0006738	-6.385423E...	-1.0013195
4933	3.0006754	-6.749579E...	-1.0013195
2407	3.0088801	0.0051216...	-1.0013195
2142	3.022922	0.0051216...	-1.0049379
1368	3.0311003	0.0051216...	-0.99105376
1328	3.0313175	0.0054259...	-0.99105376
594	3.054595	-0.0061049...	-0.99105376
560	5.0518055	-0.0061049...	-0.93253255
165	84.0443	1.7937664	0.19385245
65	85.36418	1.7937664	0.18382399

Rysunek 15: Wykres funkcji Goldsteina-Price'a z globalnym minimum: $x^* = [0.0; -1.0], f(x^*) = 3.0$

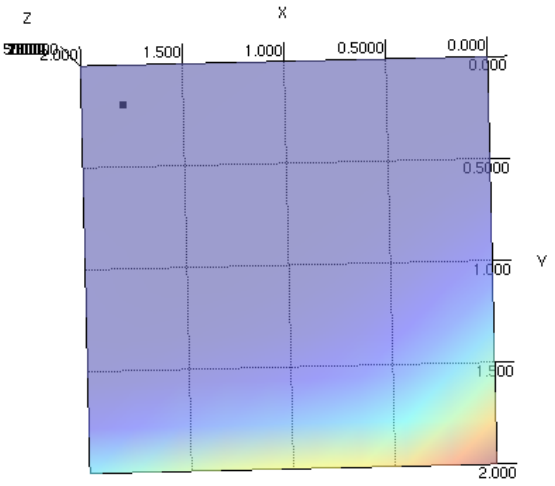


Iteracja	f(x)	x1	x2
10463	840.074	1.1982061	0.7954017
9889	840.5327	1.189241	0.7859854
9688	840.6079	1.1800063	0.7859854
9489	840.6425	1.1789744	0.7859854
1798	841.4517	1.1789744	0.776306
1629	847.08923	1.1580032	0.7471455
1625	848.11304	1.1227933	0.7471455
376	853.2524	1.1227933	0.72499895
168	853.3528	1.1161577	0.72499895
78	856.1401	1.0861757	0.72499895

Rysunek 16: Wykres funkcji Goldsteina-Price'a z lokalnym minimum: $x^*(1) = [1.2; 0.8], f[x^*(1)] = 840.0$

Rysunek 17: Wykresy funkcji Goldsteina-Price'a z lokalnymi minimami

Wykres funkcji

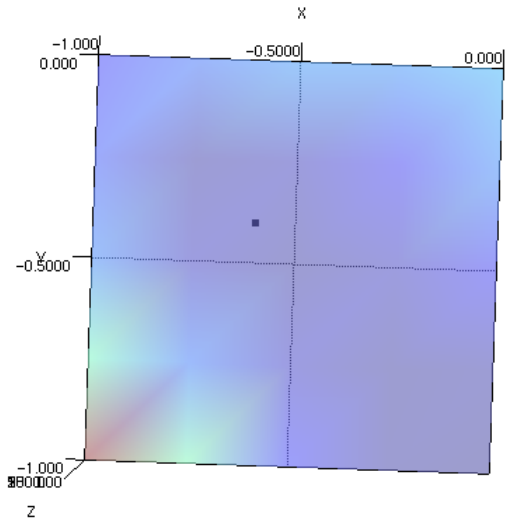


Rozwiązania

Iteracja	f(x)	x1	x2
16950	84.006966	1.7931325	0.19549912
15986	84.007	1.7931325	0.19542271
15657	84.00749	1.7929044	0.19542271
4548	84.007515	1.7929044	0.19543505
2219	84.0077	1.7929044	0.19516766
2064	84.00817	1.7929044	0.19562398
848	84.00824	1.7929044	0.19504012
288	84.04972	1.7929044	0.19749555
156	84.717995	1.7929044	0.18657683
119	88.76013	1.9550624	0.30718508

Rysunek 18: Wykres funkcji Goldsteina-Price'a z lokalnym minimum: $x^*(2) = [1.8; 0.2], f[x^*(2)] = 84.0$

Wykres funkcji

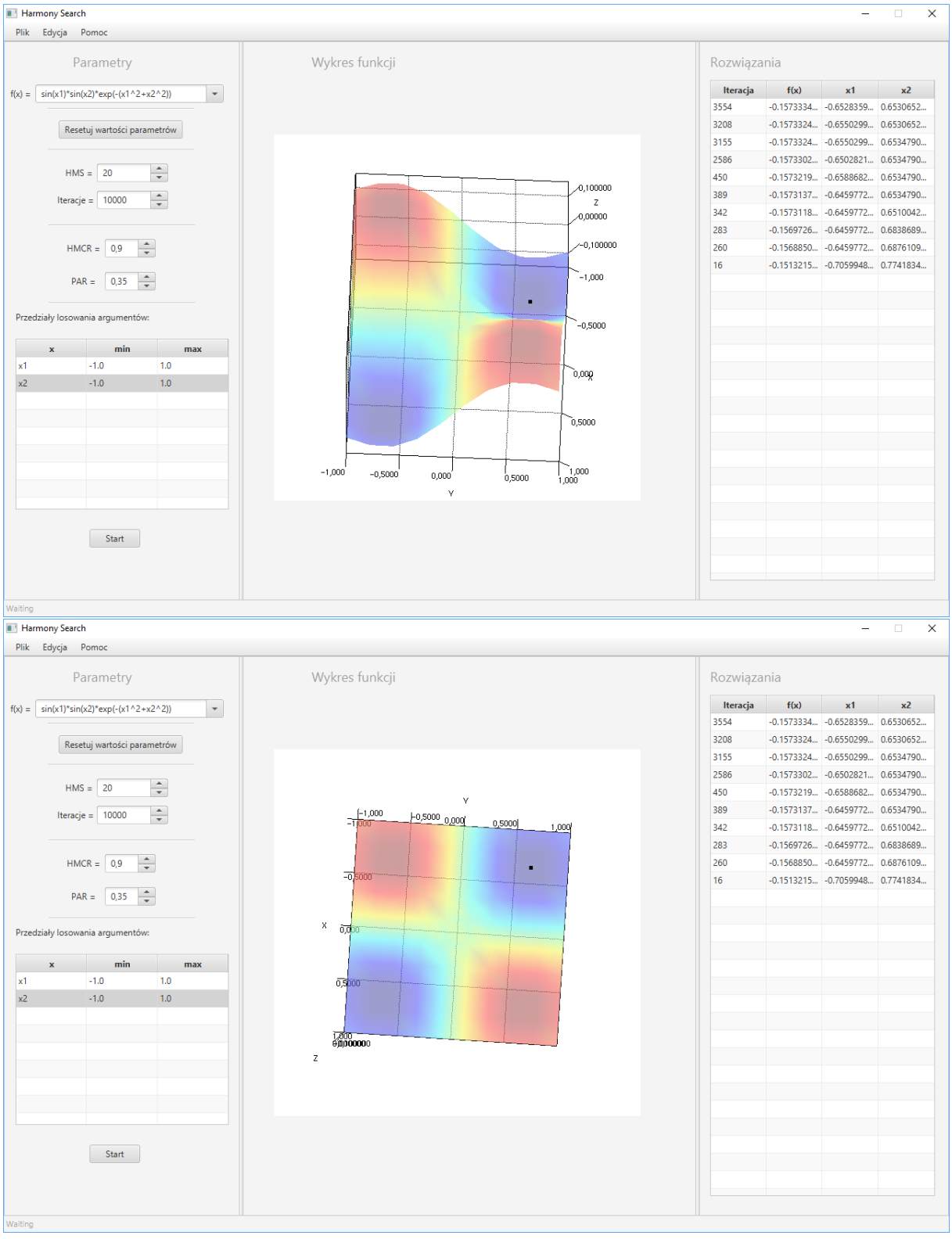


Rozwiązania

Iteracja	f(x)	x1	x2
19617	30.001791	-0.59801424	-0.402015
17103	30.001991	-0.597792	-0.402015
16801	30.002213	-0.59762853	-0.402015
16793	30.002531	-0.5974456	-0.402015
11501	30.002535	-0.5974456	-0.40194866
11156	30.00377	-0.5974456	-0.40300748
9026	30.00588	-0.5960994	-0.40300748
3038	30.006845	-0.5960994	-0.40395617
2113	30.008102	-0.59546673	-0.40395617
686	30.010622	-0.59853953	-0.40395617
323	30.130302	-0.59853953	-0.41135827
159	30.409931	-0.59853953	-0.41927904
72	30.756292	-0.56014955	-0.41927904

Rysunek 19: Wykres funkcji Goldsteina-Price'a z lokalnym minimum: $x^*(3) = [-0.6; -0.4], f[x^*(3)] = 30.0$

Rysunek 20: Wykresy funkcji Goldsteina-Price'a z lokalnymi minimami



Rysunek 21: Wykres funkcji sinusoidalnej dla $x_1, x_2 \in < -1, 1 >$

6.7 Dwuwymiarowa funkcja sinusoidalna z eksponentą

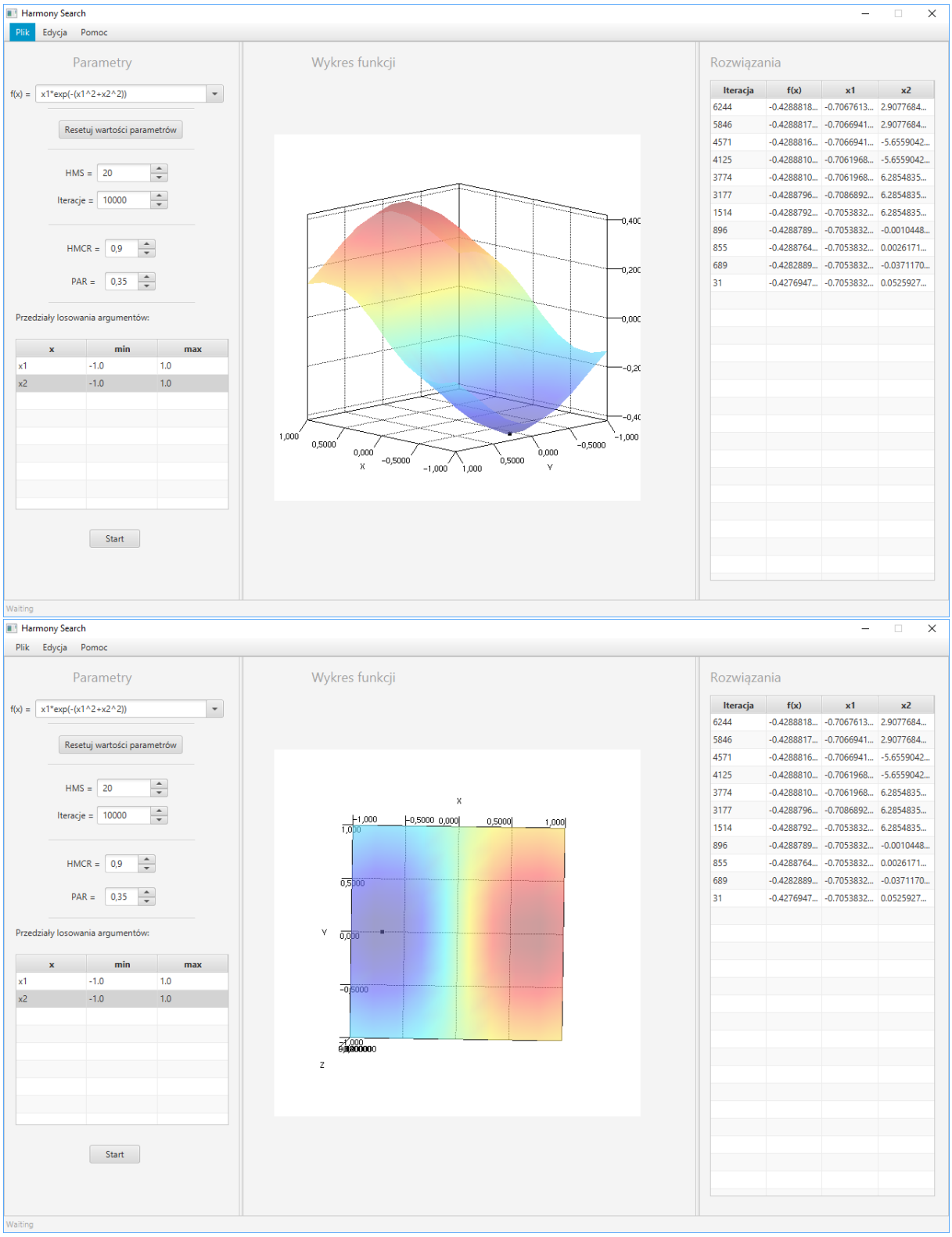
Ostatnia przedstawiana funkcja ukazuje przypadek połączenia funkcji sinusa-cosinusa z eksponentą. Przypadek jest o tyle ciekawy, że większość wartości funkcji jest dla x_1, x_2 jest do siebie zbliżona. Wzór funkcji przedstawiony jest jako:

$$f(x) = x_1 e^{-(x_1^2+x_2^2)}.$$

Rysunek 6.7 prezentuje działanie programu z obliczoną wartością minimalną. Program poprawnie wyliczył jej wartość, co potwierdza jego odporność na działanie funkcji, dla której większość argumentów jest stała.

6.8 Podsumowanie

Sprawdzając działanie programu dla różnych funkcji można stwierdzić, że poprawnie znajduje ich minima lokalne. Algorytm działa na tyle szybko, że dla użytkownika nie odczuwalny jest dyskomfort podczas użytkowania programu. Gdy funkcja posiada więcej niż jedno minimum, można znaleźć wszystkie z nich, odpowiednio kontrolując przedziały losowania argumentów lub liczbę iteracji. Można również sterować prawdopodobieństwami używanymi do generacji kolejnych rozwiązań, co może również pomóc w szybszym lub bardziej dokładnym znalezieniu szukanego minimum danej funkcji.



Rysunek 22: Wykres funkcji sinusoidalnej z eksponentą $x_1, x_2 \in < -1, 1 >$

7 Wnioski końcowe

Program obliczający minimum funkcji wielu zmiennych według algorytmu *Harmony Search* działa tak jak założono na początku. Spełniły się domniemania na temat szybkości działania algorytmu. Ponadto szybkość poprawiła się, gdy zamiast zwykłej tablicy dla `HM` przyjęto `SortedSet`. Program najdokładniej przybliży rozwiązanie, gdy zakres wartości x będzie jak najbliższy minimum funkcji. Dodatkowo, gdy ogranicznymy ilość iteracji, program zakończy szybciej swoje działanie, lecz kosztem znalezenia gorszego rozwiązania. Gdy wymagane jest obliczenie każdego z minimum funkcji możemy je znaleźć i obliczyć jego wartość, o ile znamy przedział, w którym jest ono umiejscowione. Poprzez odpowiednie ustawienie parametrów algorymu i przedziałów losowania argumentów, jesteśmy w stanie wskazać kierunek poszukiwań lokalnych optimów funkcji.

Sam algorytm Harmony Search zyskuje coraz większą popularność w praktycznych zastosowaniach [4]. Używa się do znajdywania rozwiązań problemów VRP (*ang. vehicle routing problem*, przykładowo szukanie optymalnej trasy szkolnych autobusów), projektowaniu sieci wodociągowych, systemów zapór wodnych czy pompy ciepła w satelitach. Zastosowanie znajduje także w innych dziedzinach: przewidywaniu struktury RNA, obróbce medycznych obrazów i onkologii, robotyce, śledzeniu wizualnym i wielu innych. Bardzo często daje lepsze rezultaty przy mniejszych błędach niż stosowane wcześniej metody, nie wymaga zadnych informacji o gradiencie funkcji, a także możliwe jest stosowanie do problemów dyskretnych. Wadą jest natomiast konieczność określenia parametrów działania, typowych dla algorytmów metaheurystycznych.

Literatura

[1] Kang Seok Lee, Zong Woo Geem, and G.V. Loganathan. *A New Heuristic Optimization Algorithm: Harmony Search*. 2004.

[2] Kowal Andrzej. *Algorytm Harmony Search*.

[3] Mariusz Gromada. *Dokumentacja biblioteki mXParser*. dostęp 27.05.2018. "`http://mathparser.org/`.

[4] Zong Woo Geem. *Music-Inspired Optimization Algorithm: Harmony Search*. 2004.